几乎相同图片的快速检索方法

何沧平
cangping@staff.weibo.com
微博

许涛
xutao@sugon.com
曙光信息产业（北京）有限公司

摘　要

微博平台上的大量图片几乎相同，差别仅在于水印和清晰度。为了从海量图片中找出几乎相同的图片，本文提出一种称为多层指纹的新算法。多层指纹包含 5 个字符串和 3 种实数向量，其中一个字符串用于快速召回，剩余字符串、向量用于计算指纹间相似度。多层指纹的算力需求很小，准确率高。在微博百万级图片库上测试结果表明，多层指纹的检索速度达到 QPS 345，准确率达到 97.69%。

关键词: 图片多层指纹，图片指纹，图片相似度

A fast algorithm for searching almost same images*

He Cangping
cangping@staff.weibo.com
WEIBO.COM

Xu Tao
xutao@sugon.com
Dawning Information Industry Co., Ltd

Abstract

In weibo APP, there are many almost same images whose only difference are watermark and resolution. In order to find out the most similar image efficiently, this paper proposes a algorithm named *multi-level fingerprint*, which contains 5 character strings and 3 vectors. On a dataset of 1 million images from WEIBO APP, multi-level fingerprint achieves a precision 97.69% and QPS 345.

Keywords: image fingerprint, multi-level fingerprint, image similarity

1　引言与相关工作

微博 (weibo.com) 是中国领先的媒体社交平台，海量用户每天发布数以亿计的图片。相当大比例的图片并不是原创，而是复制已有的图片，但在复制过程中会损失一些清晰度或者添加一个水印，几乎相同。为了更好地理解微博内容，需要找出来这些几乎相同的图片。

现有的图片相似度算法有两大类，传统图像处理方法，人工设计图片指纹；深度学习方法，使用训练得到的模型从图片提取一个特征向量，再用特征向量计算相似度。

*完稿日期：2020 年 10 月 10 日

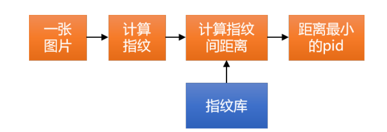


图 1: 相似图片检索流程。

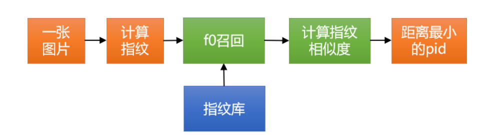


图 2: 多层指纹检索流程。

平均哈希算法（aHash）将图片缩放到指定宽高 8×8，然后计算得到一个长度为 64 的字符串。感知哈希算法（pHash）将图片缩放到指定宽高 32×32，然后计算得到一个长度为 1024 的字符串。差异哈希算法（dHash）将图片缩放到指定宽高 9×8，然后计算得到一个长度为 64 的字符串。字符串中每个字符要么是 0 要么是 1，两个字符串之间的汉明距离代表两张图片之间的相似度。MarrHildrethHash 基于 Marr-Hildreth 边缘算子计算哈希值，RadialVarianceHash 基于 Radon 变换计算哈希值，BlockMeanHash 基于块均值计算哈希值，ColorMomentHash 基于颜色矩计算哈希值。哈希类算法的详细介绍见 [9]。

SIFT 图像匹配 [5, 6]，从一张图片得到 128 维的描述子，然后匹配两个图片的关键点。SIFT 的应用场景是将同一目标在不同时间、不同分辨率、不同光照、不同位姿情况下所成的像相对应。结构相似性 (SSIM，Structural Similarity)[7] 使用的两张图像计算出一个取值范围 [0,1] 的系数。当两张图像相同时，SSIM 值等于 1。

[8] 设计一个卷积神经网络，同时输入两张图片，然后计算出相似度。工程实践中更多的方法是，用神经网络将图片转化为特征向量，具体模型非常多，最近流行有 SimCLR[2]、MoCo[4]、SwAV[1]。两个特征向量之间的距离就代表两个图片的相似度，距离通常采用余弦距离和欧式距离。

在微博场景下，图片数量很大，例如 100 亿张。如果使用传统的哈希类指纹或者神经网络类特征向量，检索相似图片需要消耗大量的算力。对照检索流程（图1），输入是一张图片，图片形式是一个三维数组，存储 RGB 灰度值。实时计算指纹模块采用某种算法由三维数组生成一个图片指 f_0，即一个定长字符串或者一个定长浮点数向量。计算指纹间距离是计算 f_0 与指纹库中所有指纹之间的距离，选择距离最小的那张图片，输出其唯一标识 pid。

“计算指纹距离”模块的资源消耗大。指纹库通常很大，存储在数据库表中，例如 mysql 表。计算 f_0 与库中指纹的距离时，通常会涉及汉明距离、欧式距离、余弦距离，但数据库的计算能力弱，计算指纹距离会消耗大量时间，导致数据支持的每秒访问次数低。高准确率和低计算量不可兼得。哈希类指纹的计算量小，但准确率低。神经网络得到特征向量准确率较高，但算法消耗多，通常要用昂贵的 GPU 支持，普通 CPU 服务器难以胜任。

为了解决算力需求大、耗时多问题，本文提出一个多层指纹算法，一张图片的多层指纹包含 5 个字符串，3 个向量。1 个字符串用于快速召回，剩余字符串和向量用于“计算指纹相似度”，见图2。使用微博搞笑领域图片测试，多层指纹的准确率达到 97.69%，检索 QPS 达到 345。

本文后续内容这样组织。第 2 节给出整图指纹，即 1 个字符串和 3 个向量的计算方法；第 3 节给出小块指纹，即 4 个字符串的计算方法；第 4 节给出多层指纹间相似度的计算方法；第 5 节给出在微博图片库上的实验结果；第 6 节总结全文。

2 整图指纹

任意给定一张图片，图片的高和宽分别记为 h 和 w，将图片的三通道值分别记为矩阵 $R=(r_{ij})_{hw}$、$G=(g_{ij})_{hw}$、$B(b_{ij})_{hw}$，矩阵的行数和列数分别为 h 和 w。显然，矩阵元素 r_{ij}、b_{ij} 和 g_{ij} 的取值范围是 $[0,255]$ 中的整数。

令

$$\bar{r}_{ij}=4\lfloor\frac{r_{ij}}{4}\rfloor,\ \bar{g}_{ij}=4\lfloor\frac{g_{ij}}{4}\rfloor,\ \bar{b}_{ij}=4\lfloor\frac{b_{ij}}{4}\rfloor,$$

这里的 $\lfloor\cdot\rfloor$ 意为向下取整。相应的灰度矩阵记为 $\bar{R}=(\bar{r}_{ij})_{hw}$，$\bar{G}=(\bar{g}_{ij})_{hw}$，$\bar{B}=(\bar{b}_{ij})_{hw}$，显然，$\bar{r}_{ij}$、$\bar{b}_{ij}$ 和 $\bar{g}_{ij}$ 的取值范围是 $0,4,8,\ldots,252$。例如，当 $r_{ij}=251$ 时，$\bar{r}_{ij}=4\lfloor\frac{251}{4}\rfloor=248$。

计算灰度比例。任意指定实数阈值 $0<\delta_1<100$，典型值 $\delta_1=1$。任意指定正整数 $3\leq m\leq 64$，典型值为 5。对 $\forall i=0,4,8,\ldots,252$，将 $\bar{R}$ 中值等于 i 的元素的数量记为 n_i，灰度 i 所占的比例记为 $u_i=\frac{100n_i}{hw}$。对 u_i 从大到小排序，截取前 m 个大于等于 δ_1 比值组成向量，即 $\boldsymbol{u}=(u_{i_{11}},u_{i_{12}},\ldots,u_{i_{1m}})$，满足条件 $u_{i_{1k}}\geq\delta_1$，$k=1,2,\ldots,m$. 如果大于等于 δ_1 的比值 $u_{i_{1k}}$ 不足 m 个，那么用 $u_{-1}=0$ 补足。例如，$\boldsymbol{u}=(u_{i_{11}},u_{i_{12}},u_{-1},u_{-1},u_{-1})=(97.09,0.531,0,0,0)$，对应的下标 $(i_{11},i_{12},-1,-1,-1)$ 值为 $(252,68,-1,-1,-1)$。

接下来调整 $\boldsymbol{u}$ 的元素的顺序。任意指定实数阈值 $\delta_2>0$，典型值 $\delta_2=0.5$。对任意两个相邻的元素 u_{i_1} 和 u_{i_2}，如果 $u_{i_1}-u_{i_2}<\delta_2$ 且 $i_1>i_2$，那么 u_{i_1} 和 u_{i_2} 互换位置。互换之后得到的向量记为 $\bar{\boldsymbol{u}}=(u_{j_{11}},u_{j_{12}},u_{j_{13}},u_{j_{14}},u_{j_5})$。例如 $\boldsymbol{u}=(7.058,6.922,5.824,4.913,4.518)$，对应下标为 $(12,16,20,24,8)$，取 $\delta_2=0.5$ 调整顺序，得到 $\bar{\boldsymbol{u}}=(7.058,6.922,5.824,4.518,4.913)$，对应下标 $(j_{11},j_{12},j_{13},j_{14})$ 的值为 $(12,16,20,8,24)$。

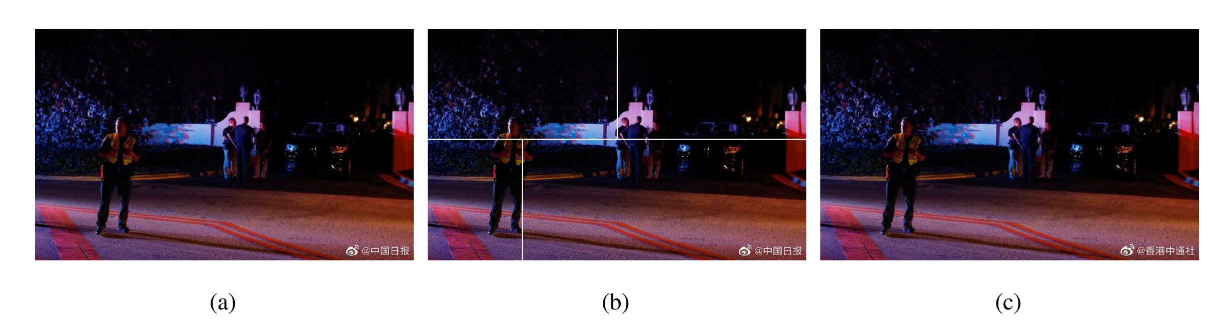
用相同的方法，对矩阵 $\bar{G}$ 计算灰度比例，得到向量 $\bar{\boldsymbol{v}}=(v_{j_{21}},v_{j_{22}},v_{j_{23}},v_{j_{24}},v_{j_{25}})$。对矩阵 $\bar{B}$ 计算灰度比例，得到向量 $\bar{\boldsymbol{z}}=(z_{j_{31}},z_{j_{32}},z_{j_{33}},z_{j_{34}},z_{j_{35}})$。如果 $\bar{\boldsymbol{u}}$、$\bar{\boldsymbol{v}}$ 和 $\bar{\boldsymbol{z}}$ 的所有元素值都是 0，那么令 $\delta_1=\frac{\delta_1}{2}$，然后重新计算 $\bar{\boldsymbol{u}}$、$\bar{\boldsymbol{v}}$ 和 $\bar{\boldsymbol{z}}$。$\bar{\boldsymbol{u}}$、$\bar{\boldsymbol{v}}$ 和 $\bar{\boldsymbol{z}}$ 称为图片的灰度向量。

拼接整图指纹 f_0。将上的数字用竖线和下划线连接成字符串，共包含 3 段。当 $m=5$ 时 f_0 = $h_w|\delta_1_\delta_2|j_{11}_j_{12}_j_{13}_j_{14}_j_{15}_j_{21}_j_{22}_j_{23}_j_{24}_j_{25}_j_{31}_j_{32}_j_{33}_j_{34}_j_{35}$，其中 _ 不是下标，是字符。例如图3(a) 的整图指纹 f_0 为

```
421_690|1.0_0.5|0_4_8_12_16_0_4_8_12_16_0_4_8_12_16
```

3 小块指纹

令 $h_1=\lfloor\frac{h}{2}\rfloor-10$，$h_2=h-h_1$，$w_1=\lfloor\frac{w}{2}\rfloor$，$w_2=w-w_1$，$w_3=\lfloor\frac{w}{4}\rfloor$，$w_4=w-w_3$。将图片分为 4 块，见图3(b)，分别称为左上、右上、左下、右下。4 块的宽高分别为 $h_1\times w_1$、$h_1\times w_2$、$h_2\times w_3$、$h_2\times w_4$。例如，图3(b) 的高宽分别为 $h=421,w=690$，那么 $h_1=200$，$h_2=221$，$w_1=345$，$w_2=345$，$w_3=172$，$w_4=518$。



(a) (b) (c)

图 3: 夜景图，来源见水印。(a) 整体，(b)4 个小块，(c) 几乎相同图。

对左上小块图片，计算其灰度向量，用灰度向量的下标拼接成字符串
$f_1 = j_{11}_j_{12}_j_{13}_j_{14}_j_{15}_j_{21}_j_{22}_j_{23}_j_{24}_j_{25}_j_{31}_j_{32}_j_{33}_j_{34}_j_{35}$
与 f_0 相比，f_1 只包含灰度值，不包含高宽、δ_1 和 δ_2。对右上小块、左下小块和右下小块作同样的操作，可得字符串 f_2、f_3、和 f_4。

将 $f_0 \sim f_4$、$\bar{\boldsymbol{u}}$、$\bar{\boldsymbol{v}}$ 和 $\bar{\boldsymbol{z}}$ 合称为图片的多层指纹。

4 图片相似度

任意给定两张图片 p^i，$i = 0, 1$，多层指纹记为 $f_0^i \sim f_4^i$、$\bar{\boldsymbol{u}}^i$、$\bar{\boldsymbol{v}}^i$ 和 $\bar{\boldsymbol{z}}^i$，两图相似度记为 s。任意指定实数阈值 $0 < \delta_3 \leq 0.2$。如果 $f_0^0 \neq f_0^1$，那么令 $s = 0$。如果 $f_0^0 = f_0^1$，对 $j = 1, 2, 3, 4$，令

$$s_j = \begin{cases} 0.2, & \text{若 } f_j^0 = f_j^1, \\ 0, & \text{若 } f_j^0 \neq f_j^1, \end{cases}$$

$$s_5 = \frac{1}{100} \max \left(\| \bar{\boldsymbol{u}}^0 - \bar{\boldsymbol{u}}^1 \|_1, \| \bar{\boldsymbol{v}}^0 - \bar{\boldsymbol{v}}^1 \|_1, \| \bar{\boldsymbol{z}}^0 - \bar{\boldsymbol{z}}^1 \|_1 \right),$$

相似度

$$s = \begin{cases} 0.2 - s_5 + \sum_{j=1}^{4} s_j, & \text{若 } s_5 < \delta_3, \\ 0, & \text{若 } s_5 \geq \delta_3. \end{cases}$$

例如，记图3(a) 为 p^0，记图3(c) 为 p^1。取 $m = 5$，$\delta_1 = 1.0$，$\delta_2 = 0.5$，$\delta_3 = 0.2$，此时 2 张图的多层指纹分别为：

f_0^0 = `421_690|1.0_0.5|0_4_8_12_16_0_4_8_12_16_0_4_8_12_16`,

f_1^0 = `0_4_8_12_16_0_4_8_12_16_4_8_0_12_16`,

f_2^0 = `0_4_8_12_16_0_4_8_12_-1_0_4_8_12_-1`,

f_3^0 = `0_4_8_12_28_0_4_8_12_16_4_0_36_8_40`,

f_4^0 = `0_4_8_112_116_0_4_8_40_44_0_4_8_24_28`,

$\bar{\boldsymbol{u}}^0 = (33.167, 12.927, 5.696, 3.042, 2.124)$,

$\bar{\boldsymbol{v}}^0 = (39.312, 12.524, 5.579, 3.16, 2.829)$,

$\bar{\boldsymbol{z}}^0 = (24.216, 14.986, 9.932, 4.325, 2.922)$,

表 1: 多层指纹的准确率

δ_3	s 最小值	正确样本数量	错误样本数量	样本总数	准确率
0.01	0.6	288	0	288	100%
0.02	0.6	325	3	328	99.09%
0.03	0.6	339	8	347	97.69%
0.2	0.6	346	31	377	91.78%

f_0^1 = `421_690|1.0_0.5|0_4_8_12_16_0_4_8_12_16_0_4_8_12_16`，
f_1^1 = `0_4_8_12_16_0_4_8_12_16_4_8_0_12_16`，
f_2^1 = `0_4_8_12_16_0_4_8_12_-1_0_4_8_12_-1`，
f_3^1 = `0_4_8_12_28_0_4_8_12_16_4_0_36_8_40`，
f_4^1 = `0_4_8_112_116_0_4_8_40_44_0_4_8_24_28`，
$\bar{\boldsymbol{u}}^1 = (33.167, 12.927, 5.696, 3.042, 2.124)$，
$\bar{\boldsymbol{v}}^1 = (39.313, 12.524, 5.578, 3.159, 2.828)$，
$\bar{\boldsymbol{z}}^1 = (24.216, 14.986, 9.931, 4.326, 2.919)$。
2 张图的实际差别只有一个水印，相似度 $s = 0.99997$ 准确地反映这个差别。

5 实验

日期 20220803-20221007 的微博图片，带概念标签的共有 1192348 张。选定参数 $m = 5$，$\delta_1 = 1$，$\delta_2 = 0.5$，计算它们的多层指纹，并存入云端 mysql 数据表。同时存入数据表的还有一个 md5 字段，图片文件的 md5 值。md5 相同的图片视为同一张图片，数据表中只保留一条记录。

对任意给定的一张图片 p_0，检索几乎相同图片的详细过程是这样。计算图片 p_0 指纹，记为 $f_0^0 \sim f_4^0$、$\bar{\boldsymbol{u}}^0$、$\bar{\boldsymbol{v}}^0$ 和 $\bar{\boldsymbol{z}}^0$。在数据表中查找与 f_0^0 完全相同的图片，这个查找操作所需计算量只有字符串比对，而且可以并行执行，速度很快。从数据表中找到 t 个图片，分别记为 p_i，$i = 1, 2, \ldots, t$，其整图指纹记为 f_0^i，$i = 1, 2, \ldots, t$，显然满足 $f_0^i = f_0^0$。然后使用第 4 节中的算法分别计算 p_0 与 p_i 的相似度，相似度最大的图片即为检索输出。

从日期 20221006-20221007 的搞笑领域中随机挑选一批图片进行检索，并人工判断正误。使用条件 $\delta_3 = 0.01$ 且 $s \geq 0.4$ 时，检索正确数量 288，检索错误数量 0，准确率 100%；使用条件 $\delta_3 = 0.02$、$\delta_3 = 0.03$、$\delta_3 = 0.2$ 时的准确率分别为 99.09%、97.69%、91.78%，详见表1。检索速度为 QPS 345。

6 总结与讨论

多层指纹的优势是算力需求低，召回速度快。计算多层指纹只涉及简单的频次统计，不像神经网络模型那样涉及大量的矩阵向量乘。整图指纹 f_0 能快速召回大致相似的图片，过滤掉绝大部分不太可能相似的图片，减少幅度可达 5 个数量级。

第 4 节中的指纹相似度，可以替换为传统的哈希指纹相似度，也可以替换为神经网络特征向量相似度，对检索速度影响不大，但可能影响检索的准确率。

参考文献

[1] Mathilde Caron et al. "Unsupervised Learning of Visual Features by Contrasting Cluster Assignments". In: *CoRR* abs/2006.09882 (2020). arXiv: 2006.09882. URL: https://arxiv.org/abs/2006.09882.

[2] Ting Chen et al. "A Simple Framework for Contrastive Learning of Visual Representations". In: (2020).

[3] Nicolas Courty, Rémi Flamary, and Mélanie Ducoffe. "Learning Wasserstein Embeddings". In: (2017). arXiv: 1710.07457. URL: https://arxiv.org/abs/1710.07457.

[4] Kaiming He et al. "Momentum Contrast for Unsupervised Visual Representation Learning". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 9726–9735. DOI: 10.1109/CVPR42600.2020.00975.

[5] David Lowe. "Distinctive Image Features from Scale-Invariant Keypoints". In: *International Journal of Computer Vision* 60 (Nov. 2004), pp. 91–. DOI: 10.1023/B:VISI.0000029664.99615.94.

[6] D.G. Lowe. "Object recognition from local scale-invariant features". In: *Proceedings of the Seventh IEEE International Conference on Computer Vision*. Vol. 2. 1999, 1150–1157 vol.2. DOI: 10.1109/ICCV.1999.790410.

[7] Zhou Wang et al. "Image quality assessment: from error visibility to structural similarity". In: *IEEE Transactions on Image Processing* 13.4 (2004), pp. 600–612. DOI: 10.1109/TIP.2003.819861.

[8] Sergey Zagoruyko and Nikos Komodakis. "Learning to Compare Image Patches via Convolutional Neural Networks". In: *CoRR* abs/1504.03641 (2015). arXiv: 1504.03641. URL: http://arxiv.org/abs/1504.03641.

[9] Christoph Zauner. "Implementation and Benchmarking of Perceptual Image Hash Functions". In: 2010.